Dylan Shadduck

EEC 201 Winter 2021

Zhi, Ding

Speaker Recognition Report

1. **Introduction**

Throughout this winter quarter we have learned a lot about different types of filters and how to manipulate signals in both the frequency and time domain. This speaker recognition project takes these fundamental concepts and applies them to a very real world problem and forces students to develop their own unique method.

1. **My Approach**

My approach for this project is most likely very similar to many other students in this class. I followed the files on Canvas for help in determining the MFCC for each speaker. This method begins by reading in each audio file using the audioread function in Matlab. This function returns a one dimensional array with all the samples and a second scalar value that represents the sampling frequency. Immediately after reading in each audio file, I normalize the audio so that the maximum amplitude for each data array is always zero. This ensures that the amplitude of the signal is not to be considered a factor when identifying the features of the speech patterns of each speaker.

The next step is to apply a trimming function to the data so that any sections without any speech are removed. This ensures that the amount of time that each person pauses before and after speaking is not identified by this program as a feature of speech. While trimming, I also tested a method of preprocessing where the data is filtered using a pre emphasis filter. After adjusting several parameters of the pre emphasis filter I found that it was not improving my results and I decided to bypass this step.

After preprocessing the next step is to perform the frame blocking method. This step takes our pre processed one dimensional array and splits it into frames of a specified length. The idea is that we will process our data in small sections so that each window we are looking at is effectively independent of each other. We also allow for a considerable amount of overlap between frames so that the edges of the frame are not lost when we perform windowing and the fft.

After frame blocking, the next step is to apply a hamming window to each frame. This can be easily done by generating a hamming window with the same length of each frame and then applying the window to each frame in the block by using a for loop.\

After we apply the hamming window to our signal, the next step is to compute the row-wise Fast Fourier Transform. The method I chose for this actually truncates the total size of our block by half because I am only concerned with keeping the positive side of the fft. This truncation is done by keeping only the first 1 + floor(N/2) terms where N is the length of our fft. Immediately after finding the fft we find the power of the fft by taking the absolute value of each element and squaring them.

Now that we have the power of the positive side of the fft computed, we can apply our mel frequency filter bank to our array. The idea behind the mel frequency filter bank is to attempt to mimic human hearing in the spacing of the filters. When humans interpret sounds we are able to perceive a linear change in frequency up to a certain threshold, but then we are only able to perceive a change in frequency on the logarithmic scale. The mel frequency filter bank mimics this design by spacing the filters in this way by following a logarithmic equation. Another important note about the mel frequency filter bank is that it tends to emphasize higher frequencies, so before applying our filter to our array we must first normalize the gain of each filter by it’s width so that all frequencies are considered equally.

The next step is to compute the Cepstrum Coefficients from our mel filtered power fft. This is done by finding the discrete cosine transform of each row in our mel filtered power fft and discarding the first row. The first row can be discarded because it contains only information about the DC component of each speaker which we don’t want to use for feature extraction.

Now that we have our Mel Frequency Filter Coefficients, we can split them up into separate clusters each with a certain centroid. The method that I chose to use for clustering is known as the kmeans algorithm. This algorithm finds k different groups in a given N dimensional dataset and returns the centroids of each group. In order to predict which speaker from the training group a given test speaker belongs to, we need to find the centroids for every single speaker in the training dataset.

Once we have the centroids for every speaker in our training dataset, we can attempt to predict who our test speaker is by fitting the mfcc’s of this speaker to each group of centroids. This process is done by finding the data point with the shortest euclidean distance to a given centroid. Once the data is fit to a kmeans model, we can evaluate the degree of fit by finding the speaker in the training set that returns the smallest possible sum euclidean distance from each mfcc point to its respective centroid. This metric tells us how closely each data point is centered around its respective cluster. We determine which speaker we predict from the training set by finding kmeans model that returns the smallest sum euclidean distance.

1. **Testing**
   1. **Test 1: Human Performance**

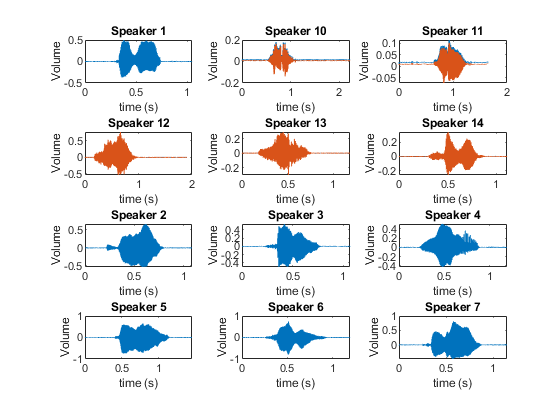
This test was done by first playing each of the training speakers several times to learn the features of each speaker. Next I had a roommate randomly play each audio file and I guessed which speaker I believed the audio sample matched. Apparently I’m not very good at identifying voices because my ability to discern the different speakers is quite poor. The table below shows the results of this test.

|  |  |
| --- | --- |
| **Ground Truth Speaker** | **Dylan’s Guess** |
| Speaker 1 | Speaker 1 |
| Speaker 2 | Speaker 2 |
| Speaker 3 | Speaker 5 |
| Speaker 4 | Speaker 4 |
| Speaker 5 | Speaker 3 |
| Speaker 6 | Speaker 7 |
| Speaker 7 | Speaker 6 |
| Speaker 8 | Speaker 4 |
| Speaker 9 | Speaker 9 |
| Speaker 10 | Speaker 10 |
| Speaker 11 | Speaker 11 |

Accuracy: 54.5%

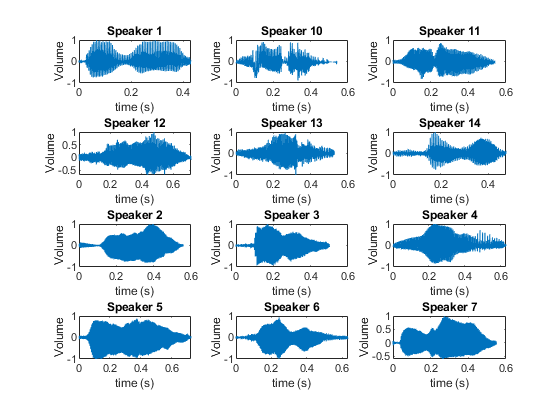
* 1. **Test 2: Time Response of Each Speaker**

In this test I was asked to plot the sampled data from each speaker vs time. The figure below shows the time domain for 12 speakers (I added my own voice to the training set) so that we can compare them.



**Figure 1: sampled audio time response**

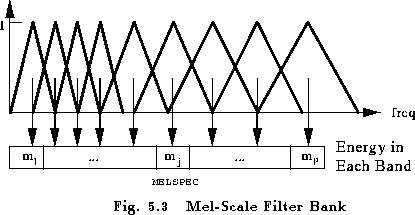
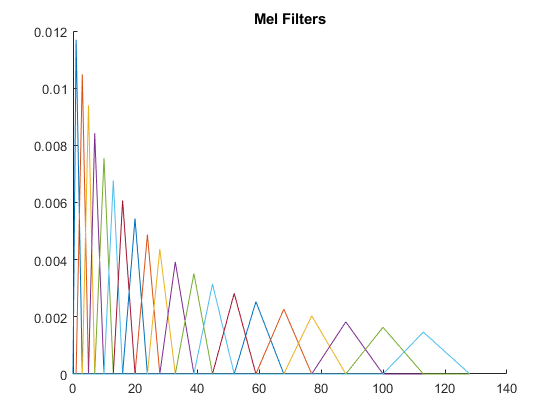
This figure shows each speaker as their speech changes over time. The graphs with orange plots have a stereo channel while the solid blue plots are only recorded with a single channel. To account for this problem as well as some other undesirable traits with all the samples, normalized all the amplitudes to one and removed the dead space before and after the speech. The figure below shows these same plots after applying the preprocessing normalization and trimming.



**Figure 2: sampled audio after preprocessing**

* 1. **Test 3: Mel Frequency Bank**

This test was to plot my mel frequency filter bank and compare it with the theoretical filter bank response. The figure below shows my generated mel frequency filter bank on the left and the theoretical filter bank on the right. I found the theoretical figure from the Laboratory for the Recognition and Organization of Speech and Audio (LabROSA) webpage.



1. **my mel filter bank b) theoretical mel filter bank**

**Figure 4: Mel Filter Bank**

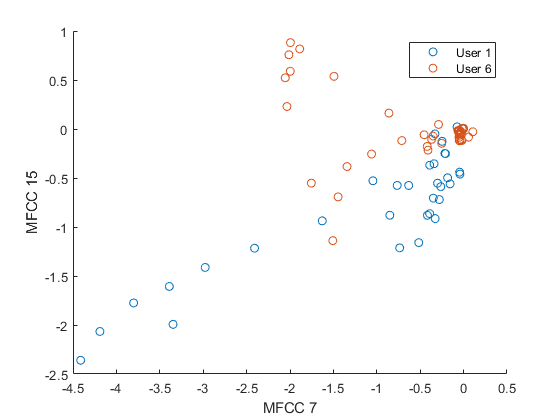
As can be seen from the figures, the spacing of the filter banks of the ones I calculated and the ones from the theoretical figure follow a similar pattern. Some key differences between them are that I chose to have much more filters, so the spacing between the filters is smaller for my plot compared to the theoretical image. Another key difference is that the amplitude of my filters is scaled such that each filter has the same area (under each triangle) so that each frequency is emphasized equally.

* 1. **Test 4: Compute MFCC in one function**

I’m not sure how this qualifies as a test, but I was able to put all the previous computation into one function with several inputs. I titled this function get\_mfcc since it extracts the mel filter cepstrum coefficients from an audio file. This function and its implementation can be found in my GitHub repository in the README document.

* 1. **Test 5: Plot 2 users in MFCC Space**

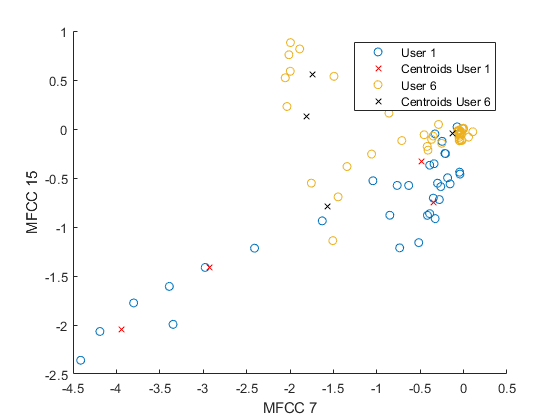
Using the get\_mfcc function that I defined I was able to plot the mel frequency cepstrum coefficients on a two dimensional scatter plot. I generate the mfcc’s using an fft length of 256, overlap length of 100, and a mel filter number of 20. The plot below shows the scatter plot for users one and six for mfcc7 vs mfcc15.



**Figure 5: MFCC visualization for users 1 and 6**

* 1. **Test 6: Plotting VQ Codewords**

In this test I first calculated the codewords (centroids) using the kmeans algorithm with 4 clusters for each speaker. The figure below shows the same plot as above, but also includes the newly calculated code words.

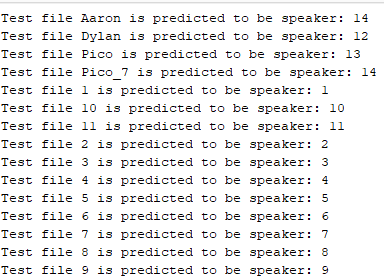


**Figure 6: MFCC visualization with codewords present**

In this figure we see that the mfcc points for users 1 and 6 remain the same, but now we can clearly see the addition of the codewords for each speaker. It does appear that the data is clustered into several groups.

* 1. **Test 7: Speaker Prediction Results**

In this test we finally get to run the full program to see if it can accurately predict which speaker is which. It is important to note that I added four new audio files to the test dataset and three new files to the training set. The new files in the training set are myself and my two roommates speaking (all say “zero”) following the same format of the other files. Of the four files added to the training set, three of them are similar to the three added to the training set, but they were recorded separately so that there may be some discrepancies. The fourth file I asked my roommate to say the number “seven”. The results of the predictions can be seen below.



**Figure 7: Predictions**

Accuracy: 93.3%

It is important to note that I labeled myself (Dylan) speaker 12, Picasso (Pico) as speaker 13, and Aaron as speaker 14. The ability of this program to predict who is speaking greatly exceeds my own abilities to discern the difference between similar speakers from test 1. The only time this program does not correctly estimate the speaker is when it is predicting the audio file when my roommate said something different than his training file. This suggests to me that to more accurately predict a certain speaker, our training data should have more of a person speaking than just a single word.

* 1. **Test 8: TODO**

1. **Conclusion**